Kris Keillor
Multi User Data Daemon (MUDD) Library
Prof. Junaid Khan
EECE 397A Wireless Networking

6/05/2022

Synopsis

The MUDD library is designed to process, filter, and tabulate data from IoT sensors. The IoT application calls upon MuddTable functions to manage data. The application may also create a TCP/IP server using MuddSocket functions, allowing multiple users to query the database. In this proof-of-concept, curated tables are stored on the IoT device and accessed via a SMB/CIFS Samba server.

Server-Client Model

The server and client communicate using the TCP/IP protocol. MUDD uses port 6545, which is unassigned according to the IANA [1]. The application must initialize the MUDD socket by calling `socket.init(port)`, and check for incoming client connections with `socket.check_readable_socket(timeout)`. When a client is detected, a separate thread is created to monitor for incoming data requests.

Clients may make requests for certain types of sensor data to be returned. These requests are automatically fulfilled by the socket threads, creating new files. The demo application creates a folder for user (thread).

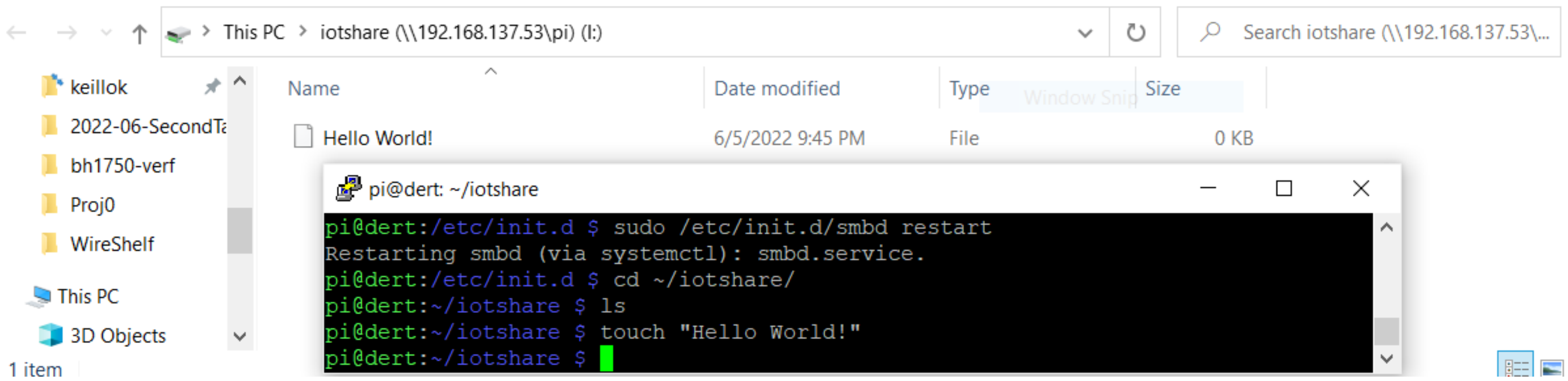
This, and the threading itself, implements the “Multi User” capabilities MUDD is named for. Running the demo as an executable script, ignoring the “hangup” signal [2], allows it to continuously run in the background as a “Daemon” server.

Demo Hardware Implementation

The MUDD Library was implemented on a Raspberry Pi Zero. Random data was generated beforehand using `table.append_entry()`. Once the demo daemon was started in the background, the SSH connection was closed.

Three clients were then opened on a Windows computer which provided WiFi to the Zero through its hotspot. The clients’ specific requests were tabulated and accessible on the Windows machine via the Raspberry Pi’s Samba share [3], as expected.

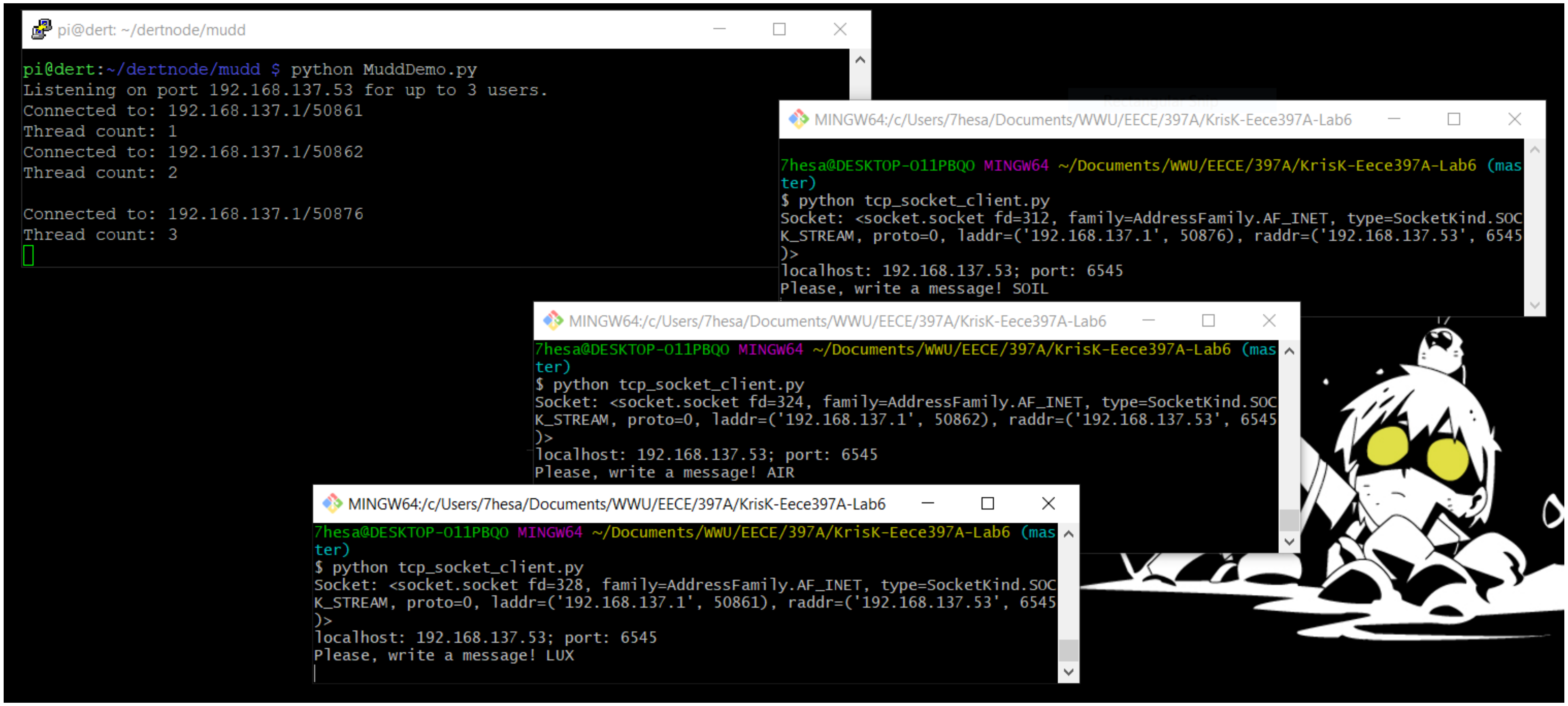
Starting the SAMBA share and creating a first file:



Name	Date modified	Type	Size
Hello World!	6/5/2022 9:45 PM	File	0 KB

```
pi@dert:/etc/init.d $ sudo /etc/init.d/smbd restart
Restarting smbd (via systemctl): smbd.service.
pi@dert:/etc/init.d $ cd ~/iotshare/
pi@dert:~/iotshare $ ls
pi@dert:~/iotshare $ touch "Hello World!"
pi@dert:~/iotshare $
```

Starting the MUD Daemon and responding to requests:



pi@dert: ~/dertnode/mudd
MINGW64:/c/Users/7hesa/Documents/WWU/EECE/397A/KrisK-Eece397A-Lab6

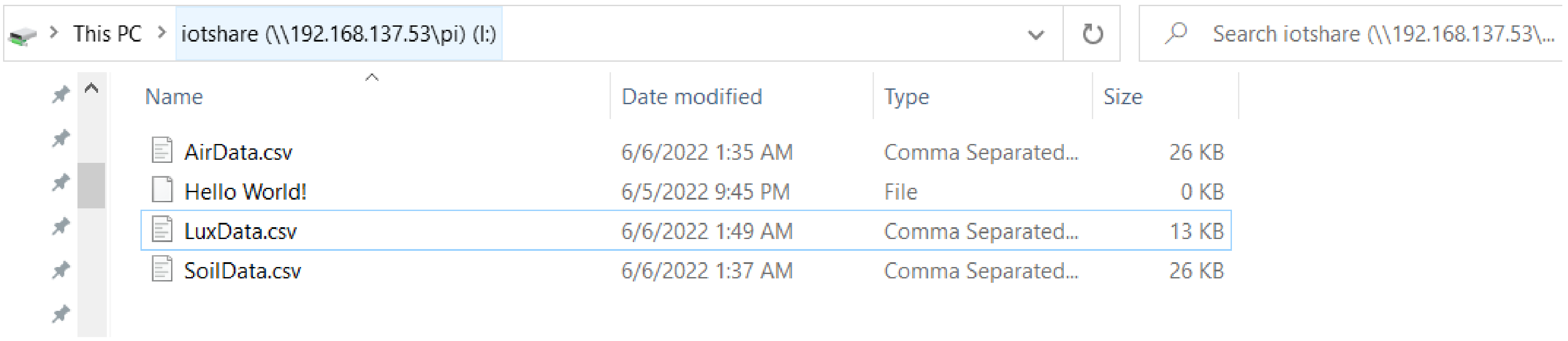
```
pi@dert:~/dertnode/mudd $ python MuddDemo.py
Listening on port 192.168.137.53 for up to 3 users.
Connected to: 192.168.137.1/50861
Thread count: 1
Connected to: 192.168.137.1/50862
Thread count: 2

Connected to: 192.168.137.1/50876
Thread count: 3
```

```
7hesa@DESKTOP-011PBQO MINGW64 ~/Documents/WWU/EECE/397A/KrisK-Eece397A-Lab6 (mas
ter)
$ python tcp_socket_client.py
Socket: <socket.socket fd=312, family=AddressFamily.AF_INET, type=SocketKind.SOC
K_STREAM, proto=0, laddr=('192.168.137.1', 50876), raddr=('192.168.137.53', 6545
)>
localhost: 192.168.137.53; port: 6545
Please, write a message! SOIL
```

```
7hesa@DESKTOP-011PBQO MINGW64 ~/Documents/WWU/EECE/397A/KrisK-Eece397A-Lab6 (mas
ter)
$ python tcp_socket_client.py
Socket: <socket.socket fd=324, family=AddressFamily.AF_INET, type=SocketKind.SOC
K_STREAM, proto=0, laddr=('192.168.137.1', 50862), raddr=('192.168.137.53', 6545
)>
localhost: 192.168.137.53; port: 6545
Please, write a message! AIR
```

```
7hesa@DESKTOP-011PBQO MINGW64 ~/Documents/WWU/EECE/397A/KrisK-Eece397A-Lab6 (mas
ter)
$ python tcp_socket_client.py
Socket: <socket.socket fd=328, family=AddressFamily.AF_INET, type=SocketKind.SOC
K_STREAM, proto=0, laddr=('192.168.137.1', 50861), raddr=('192.168.137.53', 6545
)>
localhost: 192.168.137.53; port: 6545
Please, write a message! LUX
```

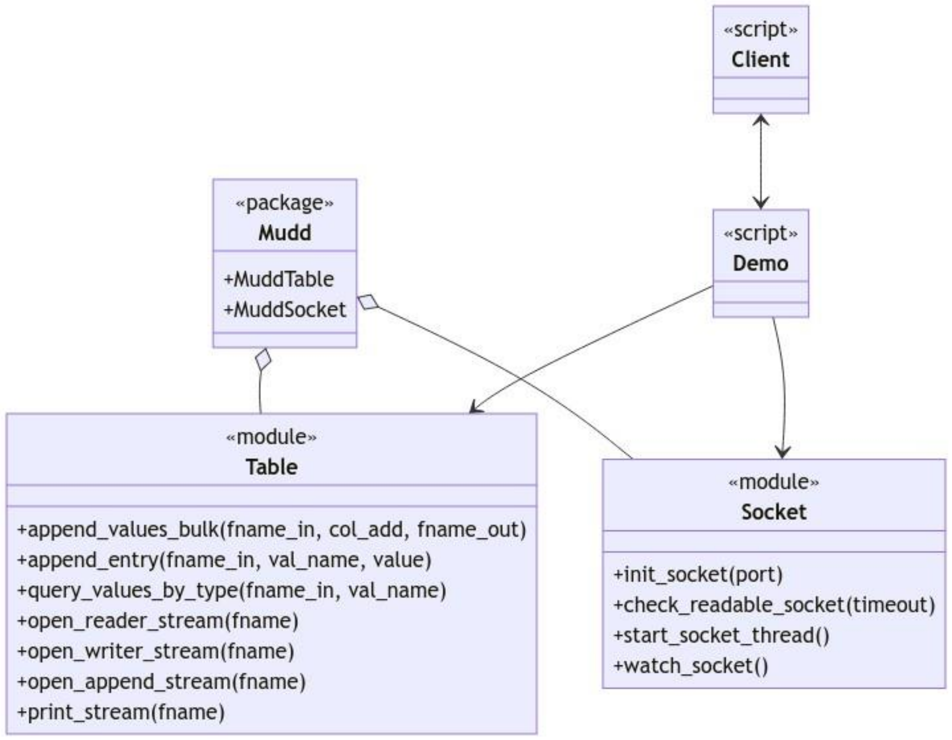
The requested data sets in the share:



This PC > iotshare (\\192.168.137.53\pi) (I:)

Name	Date modified	Type	Size
AirData.csv	6/6/2022 1:35 AM	Comma Separated...	26 KB
Hello World!	6/5/2022 9:45 PM	File	0 KB
LuxData.csv	6/6/2022 1:49 AM	Comma Separated...	13 KB
SoilData.csv	6/6/2022 1:37 AM	Comma Separated...	26 KB

Library and Demo Structure


«script»
Client
«package»
Mudd
+MuddTable
+MuddSocket
«script»
Demo
«module»
Table
+append_values_bulk(fname_in, col_add, fname_out)
+append_entry(fname_in, val_name, value)
+query_values_by_type(fname_in, val_name)
+open_reader_stream(fname)
+open_writer_stream(fname)
+open_append_stream(fname)
+print_stream(fname)
«module»
Socket
+init_socket(port)
+check_readable_socket(timeout)
+start_socket_thread()
+watch_socket()

Only `Mudd.Table` and `Mudd.Socket` are properly considered part of the library.
`Mudd.Demo`, `Mudd.DemoTable`, and `Mudd.Client` are examples to demonstrate the viability of the library. They are executed as scripts.

Conclusion

This was an extremely challenging project due to the large number of moving pieces. I chose to utilize "dummy data" so the outcomes would be safe and repeatable, which did help. I also tried to modularize the code and use exception checking, which ultimately created more problems than it solved. Notably, even when an exception was caught and handled by my try/catch statements, this masked the full traceback and made it take longer to debug. Of course there are ways around this, which I did utilize at times, but they also make things take additional time. Importing code from different files was also a headache at times.

That said, it was a very rewarding project. The MUDD library should – time permitting – allow me to access real datasets from my DERT senior project. While not a requirement of the project, it would be very interesting and allow the greenhouse to be fine-tuned and the actual conditions of the plants to be studied in greater depth.

It must be said that the project is not yet complete. The output data is not formatted correctly, and the Samba fileshare only allows the Windows user to read, not write (despite my configuration attempts). However, the networking aspects are robust.

The wireless networking aspects of this project were challenging, but most of the challenge came from using system and file calls in the context of networking. Thanks to the two prior labs, I felt comfortable using TCP/IP sockets and most of my mistakes were related to multithreading and file handling. There were, however, not-infrequent disconnects from the Raspberry Pi while SSHing in to develop and test code. This is one of the disadvantages of wireless networking compared to wired networking. Creating a file share with the SMB/CIFS protocol is a powerful ability.

References

[1]: Internet Assigned Numbers Authority. *Service Name and Transport Protocol Port Number Registry.* 03 June 2022. https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml?&page=101

[2]: Nikolai Janakiev. *Running a Python Script in the Background.* 19 Oct 2018. https://janakiev.com/blog/python-background/

[3]: Magpi Magazine. *Samba: Set up a Raspberry Pi as a File Server for your local network.* 2017. https://magpi.raspberrypi.com/articles/samba-file-server